Windows 内核模糊测试入门

Ben Nagy

COSEINC®
Solid Security.Verified.

我：

- 不是 oldsk001. 只是 old（老）.
- ~ 研究了5周的windows内核
- > 有5年模糊测试的经验
- 讨厌一切技术
- Ruby和酒精能驱散一切痛苦



免责声明:我意识到普遍的观点是模糊测试如果不谈BUG会很糟。我没有任何Bug,就算我有Bug,我也不会告诉你。没有bug,只有水獭和可疑性倾向的俄罗斯男人，还有很多红框。我警告过了哈。

COSEINC®
Solid Security.Verified.

模糊测试秘诀

- 选择正确的目标
- 必要的知识储备
- 应用各种模糊测试方法
  - 如何切入 (delivery)
  - 如何在代码中插入测试语句(Instrumentation)
  - 如何生成测试用例 (generation)
  - 怎么去扩展

COSEINC®
Solid Security. Verified.

模糊测试秘诀

- 切入,插桩, 用例生成
  - 将他们正确地分割解耦!
  - 请停止写一些耦合过高的工具, 谢谢
- 优秀工具链有助于目标的快速重定位
  - 使用糟糕的生成器开始模糊测试
  - 冰冷的心找不到Bug

COSEINC®
Solid Security.Verified.

选择目标

```
n_bugs = p_bug * n_tests
```

- **p_bug** / 目标不同测试速度也不同
- 可以平衡等式
  - 更好的（可能更慢）用例生成器
  - 更多地扩展
  - 快速使用工具 (交付时间很关键!)
  - 更好的样本
  - 预备测试工具链

COSEINC
Solid Security.Verified.

p_bug++

- 反馈驱动的模糊测试
  - 通过代码覆盖率, 成功率或者其他指标
  - 比如 SAGE, bunny, EFS, Flayer
  - 优势 – 很好, 很高级, 找到一些初级测试工具根本不会找到的Bug
  - 劣势 – 很慢开发难度大, windows的支持很差
- 错误注入 / 深度插桩模糊测试
  - 注入错误数据到被攻击代码的周围
  - 优势 – 测试切入很简单
  - 劣势 – 需要检查到达率
- 崩溃样本分析
  - 小代价高回报的技巧
  - 需要一种测度覆盖率的方法 (对于内核来说难度较大)

COSEINC®
Solid Security. Verified.

Target Selection

```
n_bugs = p_bug * n_tests
```

- 更一般地，BUG的数量没有多大意义
- 是否有更“有用”的BUG?
- 如果有，如何定位它们
  - 垃圾Bug
  - 后Fuzzing工具链

COSEINC®
Solid Security. Verified.

Target Selection

```
n_bugs = p_bug * n_tests
```

- Bug的实用性是客观存在的
- 卖掉? 应用? 修补? 散布出去?
- 无论我们的有用性定义如何，我们能收获什么？
  - 能帮助形成有用的能力?
  - 真的是可以利用的吗?
  - 有人会购买吗?
  - 值得修复吗?
  - 带来名望还是提高性能力?

COSEINC®
Solid Security.Verified.

Windows 内核简介

- Featuring “Barry the Kernel Otter”
- 有些东西丢失甚至完全错误
- 所有都过于简化了
- 资源很丰富的!
  - MSDN (新的布局和导航很赞)
  - 这几位提供的资料，j00ru, Alex Ionescu, Tarjei Mandt
  - 还有这几个人 Russinovich / Solomon / Probert
  - 学术课程“CRK”, 可免费下载
  - “WRK” ,完整Windows内核源码树,和编译工具

COSEINC®
Solid Security.Verified.

Userland

kernel32

ntdll

"NT Executive"

Dragons

Hardware

COSEINC®
Solid Security. Verified.

Userland

kernel32

ntdll

1. 建立 syscall args
2. syscall number in eax
3. int2e / sysenter / syscall

(“context switch”)

4. 在SSDT中查找syscall
5. 分发到合适的驱动

“NT Executive”

Dragons

Hardware

COSEINC®
Solid Security. Verified.

Userland

kernel32

ntdll

“NT Executive”

IO

USER

GDI

Dragons

Other Complicated Stuff

Hardware

COSEINC®
Solid Security. Verified.

Userland

kernel32

ntdll

"NT Executive"

IO

USER

GDI

Dragons

Drivers

Are

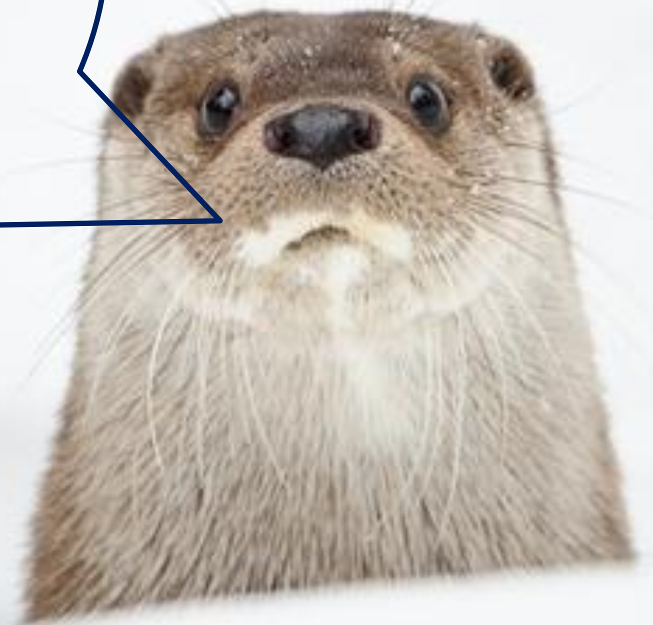
Layered!

Other Complicated Stuff

Hardware

COSEINC®
Solid Security. Verified.

- Windows IO大量使用异步操作
- 使用IO请求包（IRP）
- 设备的过滤驱动可以截获



© Sven Micklish

Userland

user32

"NT Executive"

IO

USER

GDI

Daddy Issues

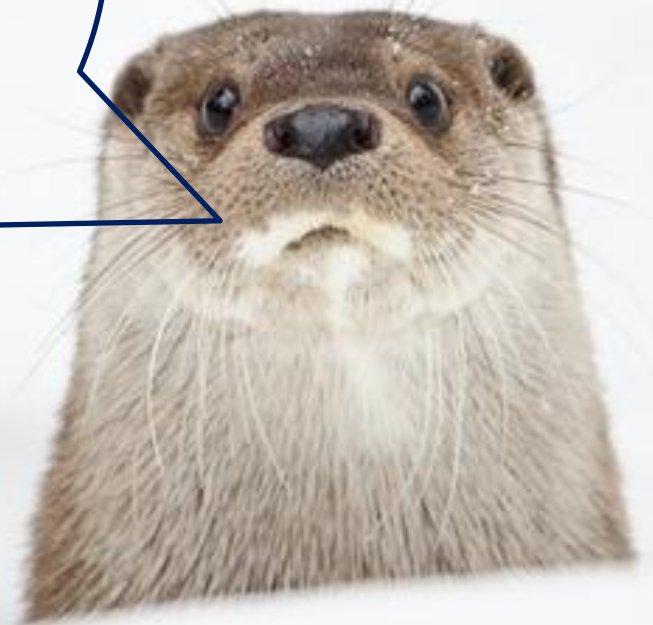
Repressed Memories

Hardware

COSEINC®
Solid Security. Verified.

用户启动界面

- 窗口, 菜单, 光标, 小图标…



© Sven Micklish

Userland

gdi32

"NT Executive"

IO

USER

GDI

Unladen Swallows

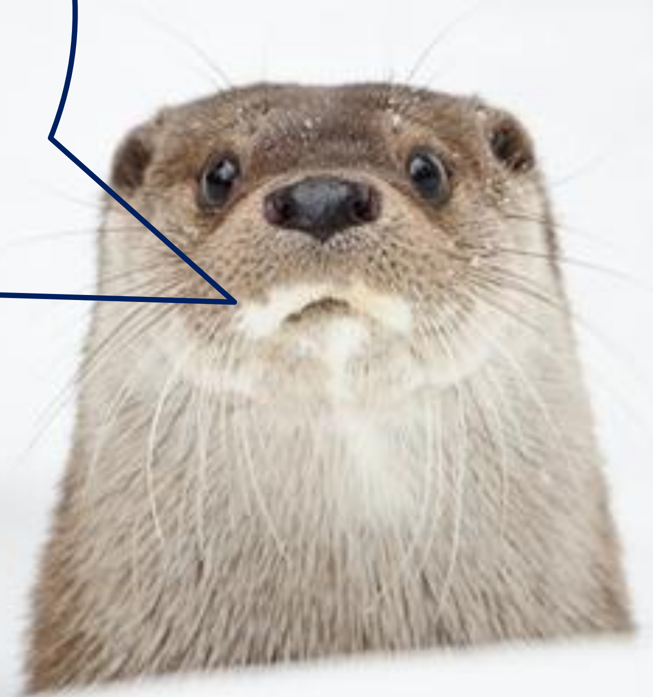
Meaning of Life

Hardware

COSEINC®
Solid Security.Verified.

图形驱动界面

- 基本上，它绘制一些东西
- 移到内核空间 ~从NT4开始
- 位图, 字体, 元信息文件...



© Sven Micklish

Userland

user32 / gdi32

"NT Executive"

IO

USER

GDI

Broccoli

Drivers

Are

Layered!

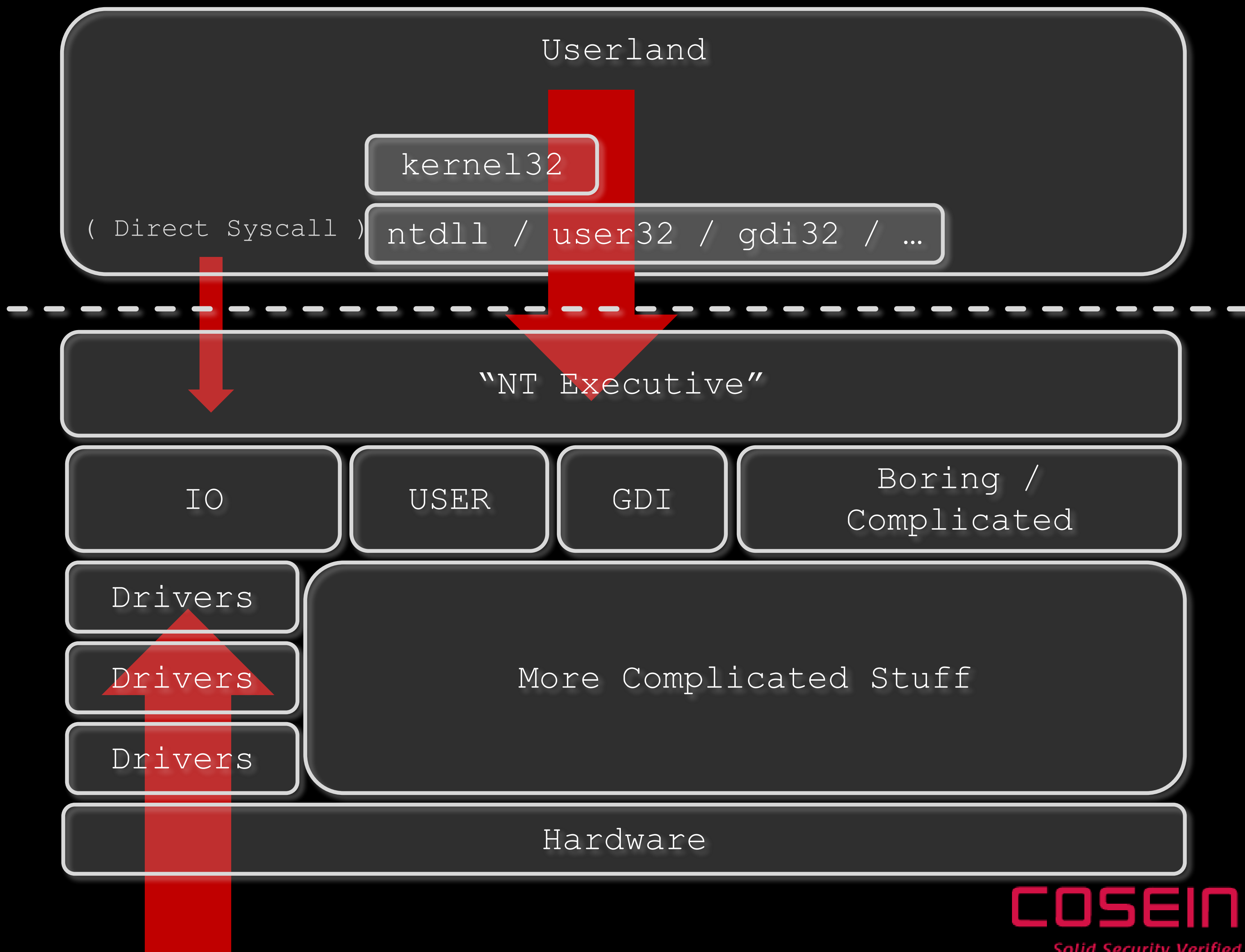
Win32k.sys

Evil Clowns

Hardware

COSEINC®
Solid Security.Verified.

Disco


Userland
kernel32
(Direct Syscall)
ntdll / user32 / gdi32 / …
"NT Executive"
IO
USER
GDI
Boring / Complicated
Drivers
Drivers
Drivers
More Complicated Stuff
Hardware
COSEINC
Solid Security.Verified.

Userland

kernel32

ntdll / user32 / gdi32 / …

Hook?

“NT Executive”

Hook?

IO

USER

GDI

Boring / Complicated

Drivers

Filter?

Drivers

Drivers

More Complicated Stuff

Hardware

COSEINC®

Solid Security. Verified.

Bug分类

- 本地到本地
  - 提权
  - 沙盒穿透
  - 趋向于更重要
- 远程到远程
  - 曾经是最好的一类BUG, 时过境迁
  - 防火墙
  - 对于无差别攻击来说很好，但是对于特定目标的攻击差一些
- 远程到本地
  - 需要用户有一些操作
  - 通过电子邮件、文档、网页地址等等进行攻击
  - 现在是Bug里面的劳斯莱斯

COSEINC®
Solid Security.Verified.

攻击方法评测

- 从硬件方法攻击
  - 实施远程到远程形式的攻击
  - 就像 ‘正常’ 网络模糊测试
  - SMB, RDP, tcpip.sys, wifi, USB…
  - 可靠性问题? 隐蔽性?
  - 硬件差别?

最后判决:你先开始，先生

COSEINC
Solid Security.Verified.

攻击方法评测

- SSDT 挂钩劫持 / 过滤驱动/ 等
  - 适合于攻击第三方驱动
  - 模糊测试的逻辑应当在内核层 (不灵活)
  - 有公开可用的实现方案
  - 比如http://code.google.com/p/ioctlfuzzer

- Finding AV bugs seems too cruel to be sport
- 不能用Ruby写驱动 ☹

COSEINC
Solid Security.Verified.

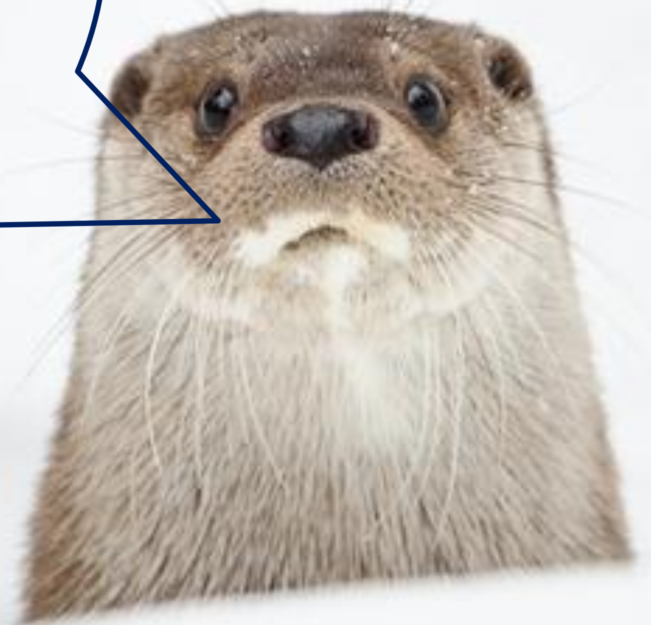
攻击方法评测

- GDI很好, 因为 RemoteLocals （远程到本地漏洞）
  - 历来就容易出Bug
- 一般系统调用 Syscalls 很有趣
  - 本地到本地, 但是建立原型比较容易
- 利用USER比较难, 只能得到一些LocalLocals（本地到本地漏洞）
  - 键盘布局被Stuxnet烧掉
  - Plus, Tarjei already looked at it

(为Bug大屠杀默哀)

COSEINC®
Solid Security.Verified.

一起搞 GDI吧!!



© Sven Micklish

GDI - Delivery Vectors

- 简单列举如下
  - 字体 - TTF, OTF, FON....
  - 光标 - BMP, CUR (animated)
  - 元数据文件 - EMF, WMF
  - 图片 - JPEG, PNG (!!)
- 不胜枚举

COSEINC®
Solid Security. Verified.

GDI – 字体

- 很好的报告演示文稿 BHEU12

http://media.blackhat.com/bh-eu-12/Lee/bh-eu-12-Lee-GDI_Font_Fuzzing-Slides.pdf

(谢谢 Lee & Chan 分享的代码)

- 字体是复杂的怪兽
- 你可以嵌入他们在浏览器里 (google搜索下 EOT)
- 只需要简单的9个步骤...

COSEINC®
Solid Security.Verified.

GDI – 字体

1. 从文件中加载模糊测试的字体

```
debug_info "Removing any old copies of #{font_file} "
GDI.RemoveFontResourceEx(font_file, 0, nil) # never
added=GDI.AddFontResourceEx(font_file, 0, nil)
```

- 我不是使用 FR_PRIVATE
- 适用于任何字体
- 高级技巧 – 修复校验和
  - （google 搜索 B1B0AFBA）

COSEINC®
Solid Security.Verified.

GDI - Window 基础

2. 建立一个Windows回调函数

```
def window_proc(hwnd, umsg, wparam, lparam)
  case umsg
    when GDI::WM_DESTROY
      GDI.PostQuitMessage(0)
      return 0
    else
      # This handles all messages we don't explicitly
process
      return GDI.DefWindowProc(hwnd, umsg, wparam,
lparam)
    end
  0
end
```

COSEINC®
Solid Security.Verified.

GDI - Window 基础

- 许多人这么认为
  - 处理 WM_PAINT, WM_RESIZE etc
  - 许多网上的样本也这么做
- 我从来不认为有这个必要, 但是见仁见智吧

COSEINC®
Solid Security. Verified.

GDI - Window 基础

3. 注册一个Window 类

```
window_class = GDI::WNDCLASSEX.new
window_class[:lpfnWndProc] =
window_class[:hInstance] = hinst
window_class[:hbrBackground] = GDI::COLOR_WINDOW
window_class[:hCursor] = 0

@atom = GDI.RegisterClassEx( window_class )
```

COSEINC®
Solid Security. Verified.

GDI - Window 基础

4. 建立一个窗口实例

```
@hwnd ||= GDI.CreateWindowEx(
  GDI::WS_EX_LEFT,                          # extended style
  poi(@atom),                                      # class name or
atom
  @opts[:title],                            # window title
  GDI::WS_OVERLAPPEDWINDOW | GDI::WS_VISIBLE, # style
  GDI::CW_USEDEFAULT,                              # X pos
  GDI::CW_USEDEFAULT,                              # Y pos
  @opts[:width],                            # width
  @opts[:height],                           # height
  0,                                        # parent
  0,                                        # menu
  hinst,                                    # instance
  nil                                       # lparam
)
```

COSEINC®
Solid Security.Verified.

GDI – 字体

5. 获取字体资源名称(未公开)

```
success=GDI.GetFontResourceInfo(
   w_fname,
   sz,
   buf,
   2 # asks to receive a LOGFONTW in buf
)
lf=LOGFONTW.new buf # cast the buffer to a
LOGFONTW
GDI.WideCharToMultiByte( …
lf[:lfFaceName].to_ptr …)
```

COSEINC®
Solid Security.Verified.

GDI – 字体

6. “建立” 该字体

```
logical_font                     = GDI::LOGFONTW.new
logical_font[:lfHeight]          = font_size
logical_font[:lfFaceName].to_ptr.put_string(0,font_face)
logical_font[:lfItalic]          = 0
logical_font[:lfCharSet] = GDI::DEFAULT_CHARSET

@current_font=GDI.CreateFontIndirect logical_font
raise_win32_error if @current_font.zero?
```

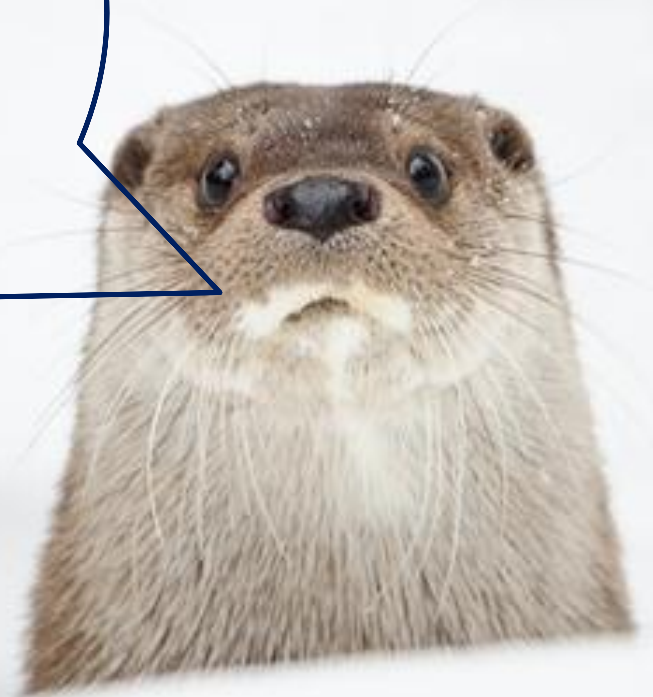
7. 将其选入已建立窗口的DC(设备环境)中

```
@old_font=GDI.SelectObject(dc, @current_font)
```

COSEINC®
Solid Security.Verified.

什么是设备环境?

- 窗口或者打印机上的信息位
- 包含各类绘图元素
- (比如 笔刷, 字体等)



© Sven Micklish

GDI – 字体

8. 一“行”文本有多大?

```
# build the string one glyph at a time until the
# text extent is greater than our rect width
sz = GDI::SIZE.new
until sz[:cx] > width || str.empty?
  out << str.slice!( 0,1 )
  GDI.GetTextExtentPoint32( dc, out, out.size,
sz )
  guess = out.size
end
```

COSEINC®

Solid Security.Verified.

GDI - Fonts

9. 绘制一些文本

```
GDI.send(
  text_out_method,                    # ExtTextOutW / A
  dc,                                 # device context
  0,                            # X start
  @current_y,                   # Y start
  GDI::ETO_GLYPH_INDEX,               # For 'raw' mode
  this_line,                          # RECT
  out,                          # str to draw
  out.size,                           # size
  nil                                 # lpDx
)
@current_y+=sz[:cy]
```

COSEINC®
Solid Security.Verified.

ETO_GLYPH_INDEX

“ lpString 数组由 GetCharacterPlacement返回并且应该被GDI直接解析，不需要进一步字体对应的语言相关的处理 ”

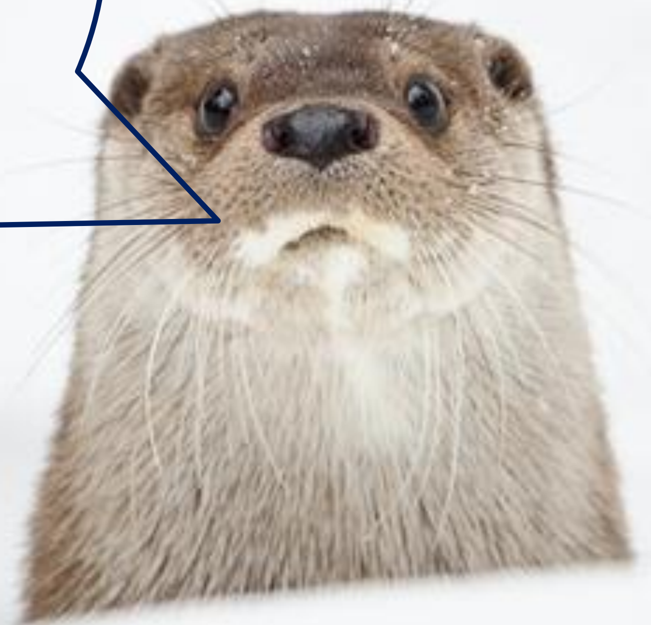
– MSDN

(这就是我们使用 ExtTextOut 而不是 DrawText 的原因)

COSEINC®
Solid Security.Verified.

糟透了!

(仍然比Gtk好一些)



© Sven Micklish



演示

Image: pavel-petel.tumblr.com - NSFW

GDI – 光标

```
hCursor=GDI.LoadCursorFromFile
raise_win32_error if hCursor.zero?
@old_cursor=GDI.SetCursor hCursor
debug_info "Set cursor #{cursor_file}”
```

- 什么意思? 为什么不是 DC?
  - 光标是共享资源!
  - 除非鼠标移动否则不应该变化
  - 还是Pff? ,随便.

COSEINC®
Solid Security.Verified.

GDI – 光标

```
@old_clip     = GDI::RECT.new
@clip     = GDI::RECT.new
GDI.SetForegroundWindow @hwnd    # _try_ to
GDI.GetClipCursor @old_clip
GDI.GetWindowRect @hwnd, @clip
GDI.ClipCursor @clip          # Clipping
GDI.ClipCursor @old_clip      # Put it back
```

- 真的很糟糕 / 脆弱的方法!
  – 尽管是可行的

COSEINC®
Solid Security.Verified.

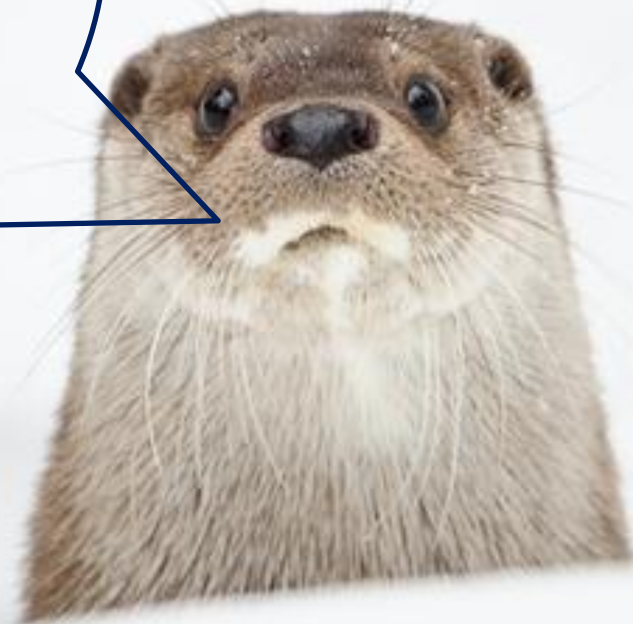


Image: pavel-petel.tumblr.com - NSFW

演示

元数据文件!

- 类似GDI命令的 ‘脚本’
- ‘可扩展’ == ‘有趣’
- SetAbortProc曾经是笑料



© Sven Micklish

GDI – 元数据文件 - WMF

```
if wmf_data[0..3] ==
  debug_info "Aldus Placeable
  pdata = pstr( wmf_data[22..-1] )
```

- WMF 文件没有大小伸缩或者位置数据
- APM 文件头是标准的 ‘非标准’
- 提供丢失的信息

COSEINC®
Solid Security.Verified.

不能伸缩! 还能做什么?

1. 在 MSPAINT.EXE 中运行
   - 内部使用 GDI+, 转换为 BMP 格式
   - 将BMP文件绘制到 DC上
2. 使用坐标空间 & 变换 API
   - 解析 APM 文件头
   - 要做很多像素和点上的繁琐的数学运算
   - 事实上就是说“像素”和“点”很讨厌
3. 转换成 EMF, 运行
   - 会丢掉一些信息，但是做起来很简单

COSEINC®
Solid Security.Verified.

GDI – 元数据文件 - WMF & EMF

```
emf_handle = GDI.SetWinMetaFileBits(
  pdata.size,
  pdata,
  dc,
  nil
) # convert to EMF if required…
raise_win32_error if emf_handle.zero?
GDI.PlayEnhMetaFile dc, emf_handle, rect
GDI.DeleteEnhMetaFile emf_handle
```

COSEINC®
Solid Security. Verified.



演示

Image: pavel-petel.tumblr.com - NSFW

COSEINC®
Solid Security.Verified.

GDI - JPEG / PNG

The **StretchDIBits** 函数复制DIB, JPEG, PNG图像文件中指定矩形区域内的颜色数据到指定的目标矩形区域。如果目标矩形区域面积大于原矩形区域，则该函数延展原区域中颜色信息的行列宽度已适应目标区域。如果目标矩形区域面积小于原矩形区域，则该函数使用指定的光栅操作压缩原区域中颜色信息的行列宽度已适应目标区域,

- MSDN

COSEINC®
Solid Security.Verified.



GDI - JPEG / PNG

为了在打印时保证合适的元数据文件缓存, 应用程序必须调用 CHECKJPEGFORMAT 或者 CHECKPNGFORMAT escape 来确信打印机在调用StretchDIBits前能够分别识别JPEG或者PNG图片文件.

- MSDN

COSEINC®
Solid Security.Verified.



好.我们来做打印机.

1. （可选）获取可选的打印机

```
buf=pstr( ″\x00″ * 260 )
buf_sz=FFI::MemoryPointer.new( :ulong )
buf_sz.write_ulong buf.size
if GDI.GetDefaultPrinter buf, buf_sz
  buf.read_string buf=pstr( ″\x00″ * 260 )
…
```

(或者只是指定 ”Fax” 等等)

COSEINC®
Solid Security.Verified.

好.我们来做打印机.

2.（可选）查看是否支持JPEG

```
escape_code=FFI::MemoryPointer.new :ulong
escape_code.write_ulong GDI::CHECKJPEGFORMAT
# Check if CHECKJPEGFORMAT exists
res=GDI.ExtEscape(
  printer_dc,
  GDI::QUERYESCSUPPORT,
  escape_code.size,
  escape_code,
  0,
  nil
)
if res > 0
  status=FFI::MemoryPointer.new :ulong
  res=GDI.ExtEscape(
    printer_dc,
    GDI::CHECKJPEGFORMAT,
    p_jpeg_data.size,
    p_jpeg_data,
    status.size,
    status
  )
```

是的，我意识到你不可能读到这些....

使用内置的打印机，比如XPS或OneNote，他们支持JPEG。

COSEINC®
Solid Security.Verified.

3. 将位图填充进结构体里

```
bmi_header                  = GDI::BITMAPINFOHEADER.new
bmi_header[:biSize]         = GDI::BITMAPINFOHEADER.size
bmi_header[:biWidth]        = img_width
# top down image - negative height value
bmi_header[:biHeight]             = -img_height
bmi_header[:biPlanes]             = 1
bmi_header[:biBitCount]           = 0
bmi_header[:biCompression]        = GDI::BI_JPEG
bmi_header[:biSizeImage]          = img_data.bytesize
```

COSEINC®
Solid Security.Verified.

4. 开始做

```
printer_dc=GDI.CreateDC nil, lpszDevice, nil, nil
retval=GDI.StretchDIBits(
  printer_dc,
  0, # dest X
  0, # dest Y
  stretch_width || rand(1000),  # width
  stretch_height || rand(1000), # height
  0, # src X
  0, # src Y
  img_width,
  img_height,
  pstr( img_data ),
  bmi_header,
  GDI::DIB_RGB_COLORS, GDI::SRCCOPY
)
```

如果返回结果大于 0，则意味着“扫描行已复制”, 并且应该和JPEG的高度是一致的。耶。

COSEINC®
Solid Security.Verified.



没有演示

Image: pavel-petel.tumblr.com - NSFW

COSEINC®
Solid Security.Verified.

还有一点…

```
# first 4 args are passed in registers.
register_args=args.shift( 4 ).zip %w( rcx rdx r8 r9 )
register_args.map! {|arg,reg| "mov #{reg}, #{arg}" }
# the rest are passed on the stack
stack_args=args.reverse.map {|arg| "push #{arg}"}
stub_x64=[
  "mov r10, rcx",                          # don't know why
  "mov eax, #{syscall}",                   # syscall in eax
  "syscall",                               # make the call
  "add rsp, #{stack_args.size * 8}",       # clean up the stack
  "ret"
]
asm = (register_args + stack_args + stub_x64).join "\n"
opcodes = Metasm::Shellcode.assemble(
  Metasm::X86_64.new, asm
).encode_string
p_opcodes = FFI::MemoryPointer.from_string opcodes
```

COSEINC®
Solid Security.Verified.

还有一点…

```
Syscall.VirtualProtect(
  p_opcodes,
  p_opcodes.size,
  PAGE_EXECUTE_READWRITE,
  FFI::MemoryPointer.new( DWORD ) # receives old protection value
)
hThread = Syscall.CreateThread(
  nil,
  0,
  p_opcodes,
  nil,
  CREATE_SUSPENDED,
  nil
)
self.raise_win32_error if hThread.zero?
Syscall.CloseHandle hThread
```

COSEINC®
Solid Security.Verified.

1行代码的Syscall模糊测试!

```
Syscall.call64
rand(0x2000),
*(Array.new(6).map {rand
2**32}) until @bsod
```

基本技术从 jduck 的 MS10-073 漏洞利用代码里借鉴的, 修改更新后可在 x86 / x64 上运行. 并且感谢Metasm团队.

COSEINC
Solid Security. Verified.

超时了!!

- 没有讲到...
- 用例生成
  - 个人主要使用 OUSPUG提供的‘Millerfuzz’ & Radamsa
  - (还有一些私密的工具)
- 扩展性
  - 通过虚拟机组对的形式扩展是很脆弱的
  - 遇到BSOD(蓝屏)时使用还原点和自动重启来应对
  - 你可以使用 NotMyFault 工具来测试
  - 分析 尚未清除的转储文件 + 还原点
  - 虚拟机重启并不能完全恢复原始状态 ☹
  - 私有的 WER 服务器会好一点儿?

COSEINC®
Solid Security.Verified.

好的，谢谢，再见



- 还是刚才提到的，5周前我对内核一无所知
- 所有我正确的看法都要归功于：
  - Lee & Chan 在 BHEU12 中的代码
  - Tarjei Mandt, Alex Ionescu, jduck
  - 新的 MSDN 导航界面
  - 运气

COSEINC®
Solid Security.Verified.

</talk>





(ben at coseinc dot com)